Examen d'algorithmique

Mercredi 5 janvier 2011 8h30/11h30 – Aucun document autorisé

Questionnaire : Pour ceux et celles qui n'ont pas rempli la fiche lors du dernier cours, un questionnaire est en ligne sur `http://www.liafa.jussieu.fr/~francoisl/l3algo.html`
Merci d'y répondre rapidement !

Mode d'emploi : Le barême est donné à titre indicatif.
Ce sujet est long : il n'est pas nécessaire de *tout* faire pour avoir une bonne note. Une bonne approche consiste à choisir 3 exercices et à les traiter correctement...
La qualité de la rédaction des algorithmes et des preuves sera très fortement prise en compte pour la note.
On peut toujours supposer une question résolue et passer à la suite...

Exercice 1 : Arbres binaires de recherche – (6 points)

On considère des arbres binaires de recherche (munis des opérations clé(a), FilsG(a), FilsD(a), et EstVide(a), et des constructeurs ABRVide( ) et ABR(x, a_1, a_2)).

Pour chacune des questions, il est possible de définir et d'utiliser d'autres fonctions que celles demandées.

1. Ecrire une fonction `Clé-Max`(a) qui retourne la clé maximale contenue dans l'ABR non-vide de racine a. Quelle est sa complexité ?
2. Ecrire une fonction `Compter-Occ`(a, x) qui retourne le nombre d'occurrences de la clé x dans l'ABR de racine a. Quelle est sa complexité ?
3. Ecrire une fonction `Prof-Moy-Occ`(a, x) qui retourne la moyenne des profondeurs des occurrences de x dans a. On suppose pour cette fonction qu'il y a toujours au moins une occurrence de x dans a.
4. Ecrire une fonction `Réduction`(a, m, M) où m et M désignent deux entiers tels que $m \leqslant M$, qui retourne un ABR contenant les clés de a appartenant à l'intervalle $[m; M]$. Quelle est sa complexité ?
5. Ecrire une fonction `Complément`(a, m, M) où m et M désignent deux entiers tels que $m \leqslant M$, qui retourne un ABR contenant les clés de a n'appartenant **pas** à l'intervalle $]m; M[$. Quelle est sa complexité ?

Exercice 2 : Arbres couvrants minimaux – (6 points)

Dans cet exercice, on considère des graphes non **orientés** et valués $G = (S, A, w)$.

On rappelle qu'un arbre couvrant minimal (ACM) de G est un sous-ensemble A' de A tel que (S, A') est connexe et acyclique (*i.e.* c'est donc un arbre) et $\sum_{(x,y)\in A'} w(x, y)$ est minimal.

Dans cet exercice, on veut calculer un ACM de G : en partant de l'ensemble d'arêtes initial A, l'algorithme devra enlever progressivement des arêtes jusqu'à obtenir un ACM.

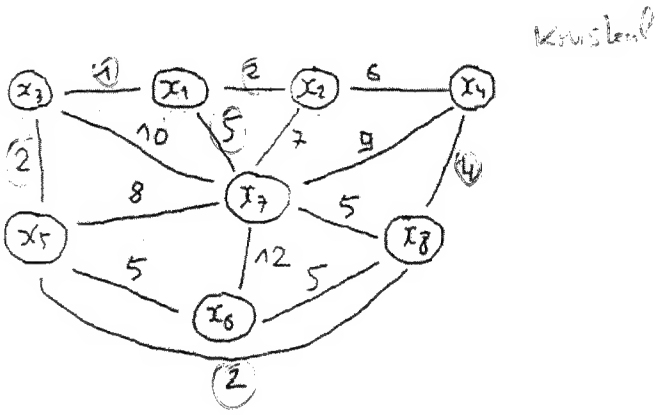

FIGURE 1 – Exemple de graphe pour l'exercice 2

1. Donner un ACM du graphe G décrit à la figure 1.
2. Ecrire une fonction `Test-Chemin`(G, x, y) qui étant donnés un graphe G et deux sommets x et y retourne *Vrai* si et seulement si il existe un chemin de x à y.
3. Etant donnés un graphe $G = (S, A, w)$ et une arête (x, y) de A, on note $G\backslash(x, y)$ le graphe G privé de (x, y), c'est-à-dire $(S, A\backslash\{(x, y)\}, w)$.
   Montrer que si G est connexe, alors on a : $G\backslash(x, y)$ est connexe **si et seulement si** il existe un chemin de x à y dans $G\backslash(x, y)$.
4. Déduire des deux questions précédentes un algorithme qui étant donné un graphe $G = (S, A, w)$ **connexe**, calcule un ACM en partant de A.
   Expliquer votre algorithme et prouver sa correction. A quelle famille d'algorithme appartient-il ? Quelle est sa complexité ? Appliquer le sur le graphe de la figure 1.

Exercice 3 : **Relation d'accessibilité – (8 points)**

On considère un graphe orienté $G = (S, A)$ avec $S = \{x_1, \ldots, x_n\}$. On note M la matrice d'adjacence de G, c'est à dire la matrice booléenne $(\alpha_{ij})_{1 \leqslant i,j \leqslant n}$ telle que : $\alpha_{ij} = \texttt{Vrai}$ si $(x_i, x_j) \in A$ et $\alpha_{ij} = \texttt{Faux}$ sinon.

On définit la somme et le produit de matrices booléennes de manière standard : la multiplication est remplacée par la conjonction ($\wedge$) et l'addition par la disjonction ($\vee$). Ainsi soient A, B et C trois matrices booléennes $n \times n$ avec les coefficients a_{ij}, b_{ij} ou c_{ij}, on a :

- $C \stackrel{\text{def}}{=} A + B$ ssi $c_{ij} = a_{ij} \vee b_{ij}$;
- $C \stackrel{\text{def}}{=} A \cdot B$ ssi $c_{ij} = \bigvee_{k=1\ldots n} (a_{ik} \wedge b_{kj})$;

On note $\texttt{Id}$ la matrice booléenne $(\gamma_{ij})_{1 \leqslant i,j \leqslant n}$ telle que $\gamma_{ii} \stackrel{\text{def}}{=} \texttt{Vrai}$ et $\gamma_{ij} \stackrel{\text{def}}{=} \texttt{Faux}$ si $i \neq j$.

On définit la suite de matrice M^k pour $k = 0, \ldots, n-1$ de la manière suivante : $M^0 \stackrel{\text{def}}{=} \texttt{Id}$ et $M^k \stackrel{\text{def}}{=} M^{k-1} \cdot M$ pour $k = 1, \ldots, n-1$.

1. Montrer que M^k, pour $k = 0, \ldots n-1$, représente la matrice d'accessibilité en exactement k transitions de G : le coefficient α^k_{ij} de M^k est `Vrai` **ssi** il existe un chemin de longueur [1] k entre x_i et x_j.
2. Soit M^* la matrice booléenne correspondant à la fermeture réflexive et transitive de la relation induite par G : le coefficient γ_{ij} de M^* est `Vrai` si et seulement si il existe un chemin (de longueur quelconque) de x_i à x_j dans G.
   Montrer que $M^* = \sum_{i=0}^{n-1} M^i$
   Quelle est la complexité de l'algorithme qui calcule M^* à partir des M^k ?
3. Adapter l'algorithme de Floyd pour calculer M^* plus efficacement.
   Donner sa complexité.
4. Etant donnés une matrice booléenne M^* représentant la fermeture réflexive et transitive de $G = (S, A)$ et un nouvel arc (x_i, x_j), donner un algorithme qui calcule la fermeture réflexive et transitive du graphe $G' = (S, A \cup \{(x, y)\})$. Donner sa complexité.

Seconde partie. On souhaite maintenant *compter* le nombre de chemins possibles entre les sommets du graphe G. Pour cela, nous allons utiliser une matrice d'adjacence N à coefficient $\alpha_{i,j}$ dans $\{0, 1\}$: $\alpha_{ij} = 1$ si $(x_i, x_j) \in A$ et $\alpha_{ij} = 0$ sinon.

On considère à présent la multiplication classique sur les matrices et on note $N^0 = \texttt{Id}$ la matrice identité (composée de 1 sur la diagonale), et $N^{k+1} \stackrel{\text{def}}{=} N^k \cdot N$. On a bien sûr $N^1 = N$.

1. Montrer que la valeur du coefficient $\alpha^k_{i,j}$ de N^k représente exactement le nombre de chemins de longueur k dans le graphe G.
2. Est-ce que les coefficients $\alpha^k_{i,j}$ ne prennent en considération que les chemins simples (sans cycle) ? Expliquer.
3. Lorsqu'on veut calculer le nombre de chemins de longueur inférieure à n entre tous les sommets de G, l'algorithme naturel se fait en $O(n^4)$. Expliquer le.
   Peut-on adapter l'algorithme de Floyd dans ce cas ? Expliquer votre réponse.

1. Dans cet exercice, la *longueur* d'un chemin est le nombre d'arcs composant ce chemin.

Exercice 4 : Rendre la monnaie – (8 points)

NB : cet exercice ne porte pas sur les graphes mais on pourra s'inspirer de la méthode utilisée dans l'algorithme de Floyd pour la question 2(b).

On souhaite construire un algorithme pour obtenir une somme S (S est un entier) à partir de $n \geqslant 1$ sortes de pièces de valeur $v_1, \ldots, v_n$. On suppose que pour chaque valeur v_i, le nombre de pièces disponibles est illimité. On suppose de plus que $v_n > v_{n-1} > \ldots > v_1$ et que $v_1 = 1$.

1. On considère l'algorithme suivant où les v_i sont stockées dans un tableau *Val*.
   Le tableau *Sol* est utilisé pour décrire une manière de composer une somme ; *Sol*[i] donne le nombre de pièces de valeur v_i. La somme correspondante est donc : $\Sigma_{i=1}^{n} Sol[i] \cdot Val[i]$.

```
Procédure Somme(S, Val)
begin
    Sol := tableau d'entiers de taille n initialisé avec[0, ..., 0];
    j := n ;
    tant que S > 0 faire
        Sol[j] := S/Val[j] ;
        S := S%Val[j] ;
        j := j − 1;
    retourner Sol
end
```

   On rappelle que / désigne la division entière et % le reste de la division entière.

   (a) Pourquoi cet algorithme termine-t-il ? A quelle famille d'algorithme appartient-il ?

   (b) Montrer qu'il fournit une solution au problème (c-à-d. une manière de composer S).

   (c) Donner le résultat renvoyé dans le cas $S = 236$ et $v_1 = 1$, $v_2 = 3$, $v_3 = 50$ et $v_4 = 70$.

2. On veut maintenant trouver un algorithme qui renvoie le **nombre minimal** de pièces pour composer la somme S (à partir des pièces de valeur $v_1, \ldots, v_n$).

   (a) Montrer sur un exemple que la solution renvoyée par l'algorithme Somme n'est pas toujours optimale de ce point de vue.

   (b) Proposer un algorithme qui résout le problème (*i.e.* donne le nombre minimal de pièces). On pourra par exemple calculer le nombre minimal $Nb[x, k]$ de pièces de valeur $v_1, \ldots, v_k$ nécessaire pour constituer la somme x, pour tout $x \leqslant S$ et $k \leqslant n$ (le nombre $Nb[S, n]$ étant alors la solution recherchée).

   (c) Adapter votre algorithme pour calculer une solution optimale (c'est-à-dire un ensemble de pièces de somme S et de taille minimale).

3. Comparer la complexité de l'algorithme Somme avec celui établi à la question précédente.

4. Comment résoudre le problème lorsque le nombre de pièces de chaque catégorie $i = 1, \ldots, n$ est limité : étant donnés une somme S, un ensemble E de pièces composé de n_1 pièces de valeur v_1, n_2 pièces de valeur v_2, *etc.*, trouver le nombre minimal de pièces nécessaires pour réaliser la somme S et correspondant à une solution faisable avec E.